Package ‘shinyGovstyle’

October 15, 2026

Title Custom Gov Style Inputs for Shiny

Version 0.3.0

Description Provides 'shiny' components styled using the GOV.UK Design System.
See <https://design-system.service.gov.uk/components/> for component guidance.

License GPL (>= 3)

URL https://github.com/dfe-analytical-services/shinyGovstyle,
https://dfe-analytical-services.github.io/shinyGovstyle/

BugReports https://github.com/dfe-analytical-services/shinyGovstyle/issues

Depends R (>= 4.1.0)

Imports bslib (>= 0.9.0), htmltools, jsonlite, lifecycle, purrr,
reactable, readODS, readr, shiny (>= 0.14), shinyjs, stringr,
writexl

Suggests brand.yml, knitr, nanonext (>= 1.8.0), pkgdown (>= 2.1.1),
rlang, rmarkdown, shinytest2, testthat

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

LazyData true

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Ross Wyatt [aut],
Cam Race [aut, cre],
Sarah Wong [aut],
Richard Bielby [ctb],
Charlotte Foster [ctb],
Jeni Martin [ctb],
Andrew Baker [ctb]

Maintainer Cam Race <camrace8@gmail.com>

Repository CRAN

Date/Publication 2026-10-15 20:40:02 UTC

Contents

accordion . . . 3
backlink_Input . . . 5
bad_link_text . . . 6
button_Input . . . 7
case_data . . . 8
checkbox_Input . . . 8
confirmation_panel . . . 10
cookieBanner . . . 12
date_Input . . . 13
details . . . 15
download_button . . . 16
download_link . . . 18
download_radios . . . 19
download_radios_handler . . . 21
error_off . . . 22
error_on . . . 23
error_summary . . . 24
error_summary_update . . . 26
external_link . . . 27
file_Input . . . 29
font . . . 31
footer . . . 32
govReactable . . . 35
govReactable-shiny . . . 37
govTable . . . 39
govTabs . . . 40
gov_list . . . 41
gov_page . . . 42
gov_summary . . . 44
gov_tag . . . 45
header . . . 46
heading_text . . . 48
input_field . . . 48
inset_text . . . 50
label_hint . . . 51
layouts . . . 53
navigate_to . . . 54
noti_banner . . . 56
phase_banner . . . 57
radio_button_Input . . . 59
run_example . . . 62
select_Input . . . 62
service_navigation . . . 64
service_navigation_server . . . 66
skip_to_main . . . 67
text_area_Input . . . 68

text_Input 70
transport_data 72
transport_data_small 72
update_page_title 73
update_radio_button_Input 74
update_service_navigation 76
value_box 78
warning_text 79

Index **80**

`accordion` *Accordion Function*

Description

This function inserts a accordion

Usage

```
accordion(inputId, titles, descriptions)
```

Arguments

`inputId`	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via `input$<inputId>`.
`titles`	Add the titles for the accordion
`descriptions`	Add the main content for each accordion section. Each item accepts a plain character string (rendered as a `govuk-body` paragraph), or `shiny` tag objects and `shiny::tagList()` values for richer block content such as multiple paragraphs, lists, or links. Tag content is inserted as-is and is *not* wrapped in a `govuk-body` paragraph, so build rich content from the styled helpers, `shinyGovstyle::gov_text()` for paragraphs and `shinyGovstyle::gov_list()` for lists, to keep GOV.UK styling. A bare tag or string passed without those helpers will render without `govuk-body` styling.

Value

an accordion HTML shiny tag object

See Also

Other Govstyle tables tabs and accordions: `govReactable()`, `govReactable-shiny`, `govTable()`, `govTabs()`

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png",
    logo_alt_text = "Ministry of Justice logo"
  ),
  shinyGovstyle::phase_banner(
    inputId = "banner", type = "beta", 'This is a new service'
  ),
  shinyGovstyle::gov_main_layout(
    accordion(
      "acc1",
      c(
        "Writing well for the web",
        "Writing well for specialists",
        "Know your audience",
        "How people read"
      ),
      list(
        "This is the content for Writing well for the web.",
        "This is the content for Writing well for specialists.",
        "This is the content for Know your audience.",
        # Rich content: a paragraph followed by a bulleted list
        # with a link
        shiny::tagList(
          shinyGovstyle::gov_text(
            "People read in different ways, including:"
          ),
          shinyGovstyle::gov_list(
            list(
              "scanning for key words",
              shiny::tags$a(
                href = "https://www.gov.uk",
                "following links"
              )
            ),
            style = "bullet"
          )
        )
      )
    )
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {}

if (interactive()) shiny::shinyApp(ui = ui, server = server)
```

backlink_Input *Back Link Function*

Description

This function adds a back link to the page

Usage

```
backlink_Input(inputId, label = "Back")
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
label	The link text for the backlink, default is "Back"

Value

a backlink HTML shiny tag object

See Also

Other Govstyle navigation: navigate_to(), service_navigation(), service_navigation_server(), update_page_title(), update_service_navigation()

Examples

```
ui <- shinyGovstyle::gov_page(
  header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png"
  ),
  shiny::navlistPanel(
    "",
    id = "nav",
    widths = c(2, 10),
    well = FALSE,
    # Create first panel
    shiny::tabPanel(
      "Select Types",
      value = "panel1",
      gov_main_layout(
        backlink_Input("link1"),
        gov_text("Placeholder text")
      )
    ),
    shiny::tabPanel(
```

```
      "Tab2",
      value = "panel2"
    )
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {
  # Slightly confused in that it goes forward rather than back
  # but shows how to use
  observeEvent(input$link1, {
    updateTabsetPanel(session, "nav", selected = "panel2")
  })
}

if (interactive()) shinyApp(ui = ui, server = server)
```

`bad_link_text` *Lookup for bad link text*

Description

A single column data frame, listing out known examples of bad link text that check for in the `external_link()` function.

Usage

```
bad_link_text
```

Format

`bad_link_text`**:**

A data frame with 53 rows and 1 columns:

bad_link_text Lower cased examples of non-descriptive link text

Details

We've started curating this list so we can create automated checks to help all link text to be as descriptive as possible in line with WCAG 2.2 success criteria 2.4.4: Link Purpose (In Context).

Source

Curated by explore.statistics@education.gov.uk

button_Input *Button Function*

Description

This function create a gov style button

Usage

```
button_Input(inputId, label, type = "default")
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
label	Display label for the control, or NULL for no label
type	The type of button. Options are default, start, secondary and warning. Defaults to "default"

Value

a HTML button shiny tag object

See Also

Other Govstyle select inputs: checkbox_Input(), file_Input(), radio_button_Input(), select_Input(), update_radio_button_Input()

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::gov_main_layout(
    shinyGovstyle::button_Input(
      inputId = "btn1",
      label = "Continue",
      type = "default"
    )
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {}

if (interactive()) shinyApp(ui = ui, server = server)
```

case_data *Case data*

Description

Simple example dataset containing case management data, used to demonstrate tabs in the example app.

Usage

```
case_data
```

Format

A data frame with 12 rows and 4 variables:

tabs The tab the data belongs to

case_manager The name of the case manager

cases_open Number of open cases

cases_closed Number of closed cases

Source

Random data generated for the shinyGovstyle package

checkbox_Input *Checkbox Function*

Description

This function inserts a checkbox group

Usage

```
checkbox_Input(
  inputId,
  cb_labels,
  checkboxIds,
  label,
  hint_label = NULL,
  small = FALSE,
  error = FALSE,
  error_message = NULL,
  label_size = c("m", "s", "l", "xl"),
  heading_level = NULL
)
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
cb_labels	Add the names of the options that will appear
checkboxIds	Add the values for each checkbox
label	Display label for the control, or NULL for no label. Accepts a plain character string, an HTML string, or shiny tag objects such as shiny::tags$b("Bold") or a shiny::tagList().
hint_label	Display hint label for the control, or NULL for no hint label. Accepts the same rich content as label, so it can include a link.
small	change the sizing to a small version of the checkbox. Defaults to FALSE
error	If TRUE, render the component in its error state and reserve a slot for the error message. Defaults to FALSE.
error_message	Text shown when error is TRUE. Defaults to NULL.
label_size	Size modifier for the legend. One of "m", "s", "l", or "xl", matching the GDS govuk-fieldset__legend--* classes. Defaults to "m".
heading_level	Optional heading level for the legend. If supplied (an integer 1-6), the legend text is wrapped in a <hN> with the GDS govuk-fieldset__heading class, following the GDS pattern for using a question as the page heading. Defaults to NULL (no heading wrap).

Value

a checkbox HTML shiny tag object

See Also

Other Govstyle select inputs: button_Input(), file_Input(), radio_button_Input(), select_Input(), update_radio_button_Input()

Examples

```
ui <- shinyGovstyle::gov_page(
  # Required for error handling function
  shinyjs::useShinyjs(),
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo="shinyGovstyle/images/moj_logo.png"),
  shinyGovstyle::phase_banner(
    inputId = "banner", type = "beta", 'This is a new service'),
  shinyGovstyle::gov_main_layout(
    # Simple checkbox
    shinyGovstyle::checkbox_Input(
      inputId = "check1",
      cb_labels = c("Option 1", "Option 2", "Option 3"),
      checkboxIds = c("op1", "op2", "op3"),
```

```
      label = "Choice option"
    ),
    # Error checkbox
    shinyGovstyle::checkbox_Input(
      inputId = "check2",
      cb_labels = c("Option 1", "Option 2", "Option 3"),
      checkboxIds = c("op1", "op2", "op3"),
      label = "Choice option",
      hint_label = "Select the best fit",
      error = TRUE,
      error_message = "Select one"
    ),
    # Rich content: a link in the hint
    shinyGovstyle::checkbox_Input(
      inputId = "check3",
      cb_labels = c("Option 1", "Option 2", "Option 3"),
      checkboxIds = c("op1", "op2", "op3"),
      label = "Choice option",
      hint_label = shiny::tagList(
        "See the ",
        shinyGovstyle::external_link(
          "https://www.gov.uk",
          "GOV.UK guidance"
        )
      )
    ),
    # Button to trigger error
    shinyGovstyle::button_Input(inputId = "submit", label = "Submit")
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {
  # Trigger error on blank submit of eventId2
  observeEvent(input$submit, {
    if (is.null(input$check2)){
      shinyGovstyle::error_on(inputId = "check2")
    } else {
      shinyGovstyle::error_off(inputId = "check2")
    }
  })
}

if (interactive()) shinyApp(ui = ui, server = server)
```

`confirmation_panel` *Confirmation panel*

Description

This function creates the GOV.UK panel component, a visible container used on confirmation or results pages to highlight important content.

Usage

```
confirmation_panel(inputId, title, content)
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
title	The panel heading, rendered as the page's <h1>.
content	The body of the panel. Accepts a plain character string, or shiny tag objects such as shiny::tags$b("Bold") or a shiny::tagList().

Value

a panel HTML shiny tag object

See Also

Other Govstyle feedback types: details(), gov_summary(), gov_tag(), inset_text(), label_hint(), noti_banner(), value_box(), warning_text()

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::gov_main_layout(
    shinyGovstyle::confirmation_panel(
      inputId = "panel1",
      title = "Application complete",
      content = paste(
        "Thank you for submitting your application.",
        "Your reference is xvsiq"
      )
    )
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {}
if (interactive()) shinyApp(ui = ui, server = server)
```

cookieBanner *Cookie Banner Function*

Description

This function creates a cookie banner. You need to have `shinyjs::useShinyjs()` enabled for this to work. All the Ids are preset. See example for how to structure.

Usage

```
cookieBanner(service_name, width = "standard")
```

Arguments

`service_name`	Name for this service to add to banner
`width`	Width of the component. One of `"standard"` (the default, GOV.UK's usual 960px content width), `"three-quarters"` (three-quarters of the viewport, never narrower than standard), `"full"` (no max-width, so the container fills the viewport, with grid gutters also removed), or a CSS length (e.g. `"1400px"`, `"90vw"`) for a custom max-width.

Value

a cookie banner HTML shiny tag object

See Also

Other Govstyle page structure: `footer()`, `gov_page()`, `header()`, `layouts`, `phase_banner()`, `skip_to_main()`

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo="shinyGovstyle/images/moj_logo.png"),
  #Needs shinyjs to work
  shinyjs::useShinyjs(),
  shinyGovstyle::cookieBanner("The best thing"),
  shinyGovstyle::gov_main_layout(),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {
  # Set of observeEvent to create a path through the cookie banner
  observeEvent(input$cookieAccept, {
    shinyjs::show(id = "cookieAcceptDiv")
    shinyjs::hide(id = "cookieMain")
```

```
  })

  observeEvent(input$cookieReject, {
    shinyjs::show(id = "cookieRejectDiv")
    shinyjs::hide(id = "cookieMain")
  })

  observeEvent(input$hideAccept, {
    shinyjs::toggle(id = "cookieDiv")
  })

  observeEvent(input$hideReject, {
    shinyjs::toggle(id = "cookieDiv")
  })

  observeEvent(input$cookieLink, {
    # Need to link here to where further info is located. You can use
    # updateTabsetPanel to have a cookie page for instance
  })

}
if (interactive()) shinyApp(ui = ui, server = server)
```

`date_Input` *Date Input Function*

Description

This function create a date input that follows GDS component

Usage

```
date_Input(
  inputId,
  label,
  hint_label = NULL,
  error = FALSE,
  error_message = NULL,
  day = NULL,
  month = NULL,
  year = NULL,
  label_size = c("m", "s", "l", "xl"),
  heading_level = NULL
)
```

Arguments

`inputId`	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via `input$<inputId>`.

label	Display label for the control, or NULL for no label. Accepts a plain character string, an HTML string, or shiny tag objects such as shiny::tags$b("Bold") or a shiny::tagList().
hint_label	Display hint label for the control, or NULL for no hint label. Accepts the same rich content as label, so it can include a link.
error	If TRUE, render the component in its error state and reserve a slot for the error message. Defaults to FALSE.
error_message	Text shown when error is TRUE. Defaults to NULL.
day	Select a default day on start up. Defaults to NULL
month	Select a default month on start up. Defaults to NULL
year	Select a default year on start up. Defaults to NULL
label_size	Size modifier for the legend. One of "m", "s", "l", or "xl", matching the GDS govuk-fieldset__legend--* classes. Defaults to "m".
heading_level	Optional heading level for the legend. If supplied (an integer 1-6), the legend text is wrapped in a <hN> with the GDS govuk-fieldset__heading class, following the GDS pattern for using a question as the page heading. Defaults to NULL (no heading wrap).

Value

a data input HTML shiny tag object

See Also

Other Govstyle text types: gov_list(), heading_text(), input_field(), text_Input(), text_area_Input()

Examples

```
ui <- shinyGovstyle::gov_page(
  # Required for error handling function.
  shinyjs::useShinyjs(),
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::phase_banner(
    inputId = "banner", type = "beta", 'This is a new service'
  ),
  shinyGovstyle::gov_main_layout(
    # Simple date input
    shinyGovstyle::date_Input(
      inputId = "dob_input",
      label = "Please enter your birthday"
    ),
    # Error date input
    shinyGovstyle::date_Input(
      inputId = "dob_input2",
```

```
        label = "Please enter your birthday",
        hint_label = "For example, 12 11 2007",
        error = TRUE
      ),
      # Rich content: a link in the hint
      shinyGovstyle::date_Input(
        inputId = "dob_input3",
        label = "Please enter your birthday",
        hint_label = shiny::tagList(
          "Check the format in the ",
          shinyGovstyle::external_link(
            "https://www.gov.uk",
            "GOV.UK guidance"
          )
        )
      ),
      # Button to trigger error
      shinyGovstyle::button_Input(inputId = "submit", label = "Submit")
    ),
    shinyGovstyle::footer(full = TRUE)
  )

  server <- function(input, output, session) {
    # Trigger error on blank submit of dob_input2
    observeEvent(input$submit, {
      if (input$dob_input2 == "//") {
        shinyGovstyle::error_on(inputId = "dob_input2")
      } else {
        shinyGovstyle::error_off(inputId = "dob_input2")
      }
    })
  }

  if (interactive()) shinyApp(ui = ui, server = server)
```

`details` *Details Function*

Description

This function create a detail component that you can click for further details.

Usage

```
details(inputId, label, help_text)
```

Arguments

`inputId`	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via `input$<inputId>`.

label	Main label text
help_text	Additional help information in the component. Accepts a plain character string, or shiny tag objects such as shiny::tags$b("Bold") or a shiny::tagList().

Value

a details box HTML shiny tag object

See Also

Other Govstyle feedback types: confirmation_panel(), gov_summary(), gov_tag(), inset_text(), label_hint(), noti_banner(), value_box(), warning_text()

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::gov_main_layout(
    shinyGovstyle::details(
      inputId = "help_div",
      label = "Help with form",
      help_text = "To complete the form you need to fill it in..."
    )
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {}

if (interactive()) shinyApp(ui = ui, server = server)
```

---

download_button *Download button*

Description

The download_button() provides a standard way to provide a download link, which facilitates important accessible / positive user experience elements, namely:

- file type
- file size

These are necessary in order for users to understand what they are downloading, both in terms of being able to decide if they are comfortable with downloading the file over their current connection and if it's in a form they're able to deal with once it is downloaded. If the exact file size is not easily determined, then it may be acceptable to provide an estimate or an upper limit.

Usage

```
download_button(outputId, button_label, file_type = "CSV", file_size = NULL)
```

Arguments

outputId	The name of the output slot that the shiny::downloadHandler() is assigned to.
button_label	Text that will appear describing the download action. Vague text like 'click here' or 'here' will cause an error, as will ending in a full stop. Leading and trailing white space will be automatically trimmed. If the string is shorter than 7 characters a console warning will be thrown. There is no way to hush this other than providing more detail
file_type	File extension shown to the user (e.g. "CSV", "PDF"). Defaults to "CSV".
file_size	Optional human-readable file size; a string ending in KB, MB, GB, or rows.

Value

shiny tag object

See Also

Other Govstyle actions: download_link(), download_radios(), download_radios_handler(), external_link()

Examples

```
ui <- shinyGovstyle::gov_page(
  gov_text("Choose a data set to download."),
  select_Input(
    "dataset",
    "Data set",
    select_text = c("Car road tests", "New York air quality"),
    select_value = c("mtcars", "airquality")
  ),
  download_button(
    "download_data",
    "Download selected data set",
    file_size = "4 KB"
  )
)

server <- function(input, output) {
  # The requested data set
  data <- reactive({
    get(input$dataset)
  })

  output$download_data <- downloadHandler(
    filename = function() {
      # Use the selected dataset as the suggested file name
```

```
      paste0(input$dataset, ".csv")
    },
    content = function(file) {
      # Write the dataset to the `file` that will be downloaded
      write.csv(data(), file)
    }
  )
}

if (interactive()) shiny::shinyApp(ui, server)
```

---

download_link *Download link*

Description

The `download_link()` provides a standard way to provide a download link, which facilitates important accessible / positive user experience elements, namely:

- file type
- file size

These are necessary in order for users to understand what they are downloading, both in terms of being able to decide if they are comfortable with downloading the file over their current connection and if it's in a form they're able to deal with once it is downloaded. If the exact file size is not easily determined, then it may be acceptable to provide an estimate or an upper limit.

Usage

```
download_link(outputId, link_text, file_type = "CSV", file_size = NULL)
```

Arguments

`outputId`	The name of the output slot that the `shiny::downloadHandler()` is assigned to.
`link_text`	Text that will appear describing the download action. Vague text like 'click here' or 'here' will cause an error, as will ending in a full stop. Leading and trailing white space will be automatically trimmed. If the string is shorter than 7 characters a console warning will be thrown. There is no way to hush this other than providing more detail
`file_type`	File extension shown to the user (e.g. `"CSV"`, `"PDF"`). Defaults to `"CSV"`.
`file_size`	Optional human-readable file size; a string ending in `KB`, `MB`, `GB`, or `rows`.

Value

shiny tag object

See Also

Other Govstyle actions: download_button(), download_radios(), download_radios_handler(), external_link()

Examples

```
ui <- shinyGovstyle::gov_page(
  gov_text("Choose a data set to download."),
  select_Input(
    "dataset",
    "Data set",
    select_text = c("Car road tests", "New York air quality"),
    select_value = c("mtcars", "airquality")
  ),
  gov_text(
    download_link(
      "download_data",
      "Download selected data set",
      file_size = "4 KB"
    )
  )
)

server <- function(input, output) {
  # The requested data set
  data <- reactive({
    get(input$dataset)
  })

  output$download_data <- downloadHandler(
    filename = function() {
      # Use the selected dataset as the suggested file name
      paste0(input$dataset, ".csv")
    },
    content = function(file) {
      # Write the dataset to the `file` that will be downloaded
      write.csv(data(), file)
    }
  )
}

if (interactive()) shiny::shinyApp(ui, server)
```

---

download_radios *Download with extension radios*

Description

Download with extension radios

Usage

```
download_radios(
  id = "download_radios",
  download_type = "table",
  file_types = c("CSV", "ODS", "XLSX"),
  file_sizes = c("< 1 GB", "< 1 GB", "< 1 GB"),
  small = FALSE
)
```

Arguments

id	Shiny element Id. Default is "download_radios", but must be customised to be unique if multiple instances of this module are being used in a single app. Must match up to the ID of a `download_radios_helper()` instance in server code
download_type	Element being downloaded. Expected to be along the lines of "underlying data", "table", "current data view". Default is "table"
file_types	File formats to offer, can be any combination of "CSV", "XLSX", "ODS". Default is a vector of all 3
file_sizes	Estimated file sizes for each file format. This needs to be a vector of the same length as file_types
small	Set radio buttons to small size (logical, default: FALSE)

Value

UI containing radio selection and download button

See Also

Other Govstyle actions: `download_button()`, `download_link()`, `download_radios_handler()`, `external_link()`

Examples

```
ui <- shinyGovstyle::gov_page(
  download_radios("download_file",
    file_types = c("CSV", "ODS"),
    file_sizes = c("2 KB", "5 KB")
  )
)

server <- function(input, output, session) {
  output$download_file <- download_radios_handler(
    "download_file",
    file_name = "simple_data_frame",
    file_contents = mtcars
  )
}

if (interactive()) shinyApp(ui, server)
```

download_radios_handler

Download with extension radios handler

Description

Download with extension radios handler

Usage

```
download_radios_handler(id = "download_radios", file_name, file_contents)
```

Arguments

id	Shiny element Id. Default is "download_radios", but must be customised to be unique if multiple instances of this module are being used in a single app. Must match up to the ID of a download_radios() instance in UI code
file_name	Name of the file to be downloaded
file_contents	Contents to write to the download file

Value

Output for use with download_radios()

See Also

Other Govstyle actions: download_button(), download_link(), download_radios(), external_link()

Examples

```
ui <- shinyGovstyle::gov_page(
  download_radios("download_file",
    file_types = c("CSV", "ODS"),
    file_sizes = c("2 KB", "5 KB")
  )
)

server <- function(input, output, session) {
  output$download_file <- download_radios_handler(
    "download_file",
    file_name = "simple_data_frame",
    file_contents = mtcars
  )
}

if (interactive()) shinyApp(ui, server)
```

error_off *Error off Function*

Description

This function turns off the the error o the component, once issues have been sorted.

Usage

```
error_off(inputId)
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.

Value

no return value. This toggles off error CSS

See Also

Other Govstyle errors: error_on(), error_summary(), error_summary_update()

Examples

```
ui <- shinyGovstyle::gov_page(
  # Required for error handling function
  shinyjs::useShinyjs(),
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::phase_banner(
    inputId = "banner", type = "beta", 'This is a new service'
  ),
  shinyGovstyle::gov_main_layout(
    # Error text box
    shinyGovstyle::text_Input(
      inputId = "eventId",
      label = "Event Name",
      error = TRUE
    ),
    # Button to trigger error
    shinyGovstyle::button_Input(inputId = "submit", label = "Submit")
  ),
  shinyGovstyle::footer(full = TRUE)
)
```

```
server <- function(input, output, session) {
  # Trigger error on blank submit of eventId2
  observeEvent(input$submit, {
    if (input$eventId != "") {
      shinyGovstyle::error_off(inputId = "eventId")
    } else {
      shinyGovstyle::error_on(
        inputId = "eventId",
        error_message = "Please complete"
      )
    }
  })
}

if (interactive()) shinyApp(ui = ui, server = server)
```

---

error_on *Error on Function*

Description

This function turns on the error for the component. Can be used to validate inputs.

Usage

```
error_on(inputId, error_message = NULL)
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
error_message	if you want to add an additional error message Defaults to NULL, showing the original designed error message. Plain character strings are escaped and render as literal text; pass a shiny tag, shiny::tagList(), or shiny::HTML() to render markup.

Value

no return value. This toggles on error CSS

See Also

Other Govstyle errors: error_off(), error_summary(), error_summary_update()

Examples

```
ui <- shinyGovstyle::gov_page(
  # Required for error handling function
  shinyjs::useShinyjs(),
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo="shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::phase_banner(
    inputId = "banner", type = "beta", 'This is a new service'
  ),
  shinyGovstyle::gov_main_layout(
    # Error text box
    shinyGovstyle::text_Input(
      inputId = "eventId",
      label = "Event Name",
      error = TRUE),
    # Button to trigger error
    shinyGovstyle::button_Input(inputId = "submit", label = "Submit")
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {
  # Trigger error on blank submit of eventId2
  observeEvent(input$submit, {
    if (input$eventId != ""){
      shinyGovstyle::error_off(inputId = "eventId")
    } else {
      shinyGovstyle::error_on(
        inputId = "eventId",
        error_message = "Please complete"
      )
    }
  })
}

if (interactive()) shinyApp(ui = ui, server = server)
```

`error_summary` *Error Summary Function*

Description

This function loads the error summary component to display error text. This replicates the gov style error boxes linked below: https://design-system.service.gov.uk/components/error-summary/

Usage

```
error_summary(inputId, error_title, error_list)
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
error_title	The title for the error summary
error_list	A list of text values to be displayed in the error body

Value

an error_summary HTML shiny tag object

See Also

Other Govstyle errors: error_off(), error_on(), error_summary_update()

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyjs::useShinyjs(),
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::gov_main_layout(
    error_summary(
      inputId = "errorId",
      error_title = "Error title",
      error_list = c("error item1", "error item2")
    )
  ),
  shinyGovstyle::button_Input("btn1", "Change error summary"),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {
  shiny::observeEvent(input$btn1, {
    error_summary_update(
      "errorId",
      c("error item1", "error item2", "error item3")
    )},
    ignoreInit = TRUE
  )
}

if (interactive()) shinyApp(ui = ui, server = server)
```

error_summary_update *Error Summary Update Function*

Description

This function changes the text that displays in the error summary box. Requires `shinyjs::useShinyjs()` to work.

Usage

```
error_summary_update(inputId, error_list)
```

Arguments

`inputId`	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via `input$<inputId>`.
`error_list`	An updated list of text values to be displayed in the error body

Value

an update error summary box

See Also

Other Govstyle errors: `error_off()`, `error_on()`, `error_summary()`

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyjs::useShinyjs(),
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::gov_main_layout(
    error_summary(
      inputId = "errorId",
      error_title = "Error title",
      error_list = c("error item1", "error item2")
    )
  ),
  shinyGovstyle::button_Input("btn1", "Change error summary"),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {
  shiny::observeEvent(input$btn1, {
    error_summary_update(
```

```
        "errorId",
        c("error item1", "error item2", "error item3")
      )
    },
    ignoreInit = TRUE
    )
  }

  if (interactive()) shinyApp(ui = ui, server = server)
```

`external_link` *External link*

Description

It is commonplace for external links to open in a new tab, and when we do this we should be careful...

This function automatically adds the following to your link:

- `target="_blank"` to open in new tab
- `rel="noopener noreferrer"` to prevent reverse tabnabbing

By default this function also adds "(opens in new tab)" to your link text to warn users of the behaviour.

This also adds "This link opens in a new tab" as a visually hidden span element within the HTML outputted to warn non-visual users of the behaviour.

The function will error if you end with a full stop, give a warning for particularly short link text and will automatically trim any leading or trailing white space inputted into link_text.

If you are displaying lots of links together and want to save space by avoiding repeating (opens in new tab), then you can set add_warning = FALSE and add a line of text above all of the links saying something like 'The following links open in a new tab'.

Setting add_warning = FALSE removes the visible text warning entirely, so sighted users lose the visual cue that the link behaves differently. Set add_warning = "icon" instead to add a small decorative arrow icon after the link text, giving sighted users a visual warning without repeating the full "(opens in new tab)" text. The icon is purely decorative (hidden from screen readers, which still get the same hidden warning as add_warning = FALSE), so it should be paired with an explanatory sentence above a group of links, not relied on as the only warning.

If `href` starts with `mailto:` (case-insensitive), the new-tab behaviour is skipped entirely: no `target="_blank"`/`rel="noopener noreferrer"`, no "(opens in new tab)" text or hidden span, and `add_warning = "icon"` has no effect, since a mailto link hands off to the mail client rather than opening a new browser tab. `link_text` validation still applies as normal.

Related links and guidance:

- Government digital services guidelines on the use of links
- Anchor tag HTML element and its properties
- WCAG 2.2 success criteria 2.4.4: Link Purpose (In Context)
- Web Accessibility standards link text behaviour

Usage

```
external_link(href, link_text, add_warning = TRUE, footer = FALSE)
```

Arguments

`href`	URL that you want the link to point to
`link_text`	Text that will appear describing your link, must be descriptive of the page you are linking to. Vague text like 'click here' or 'here' will cause an error, as will ending in a full stop. Leading and trailing white space will be automatically trimmed. If the string is shorter than 7 characters a console warning will be thrown. There is no way to hush this other than providing more detail
`add_warning`	One of TRUE (default), FALSE, or "icon", controlling how users are warned that the link opens in a new tab. TRUE appends "(opens in new tab)" to the visible link text. FALSE removes the visible text (a visually hidden span still warns screen reader users) with no replacement visual cue, for use with an explanatory sentence above a group of links. "icon" behaves like FALSE but also adds a small decorative arrow icon after the link text, giving sighted users a visual cue without repeating the full text. Be careful and consider accessibility before moving away from the default. Has no effect when `href` is a `mailto:` link.
`footer`	Apply standard GDS footer CSS styling. Logical, default = FALSE

Details

Intentionally basic wrapper for HTML anchor elements making it easier to create safe external links with standard and accessible behaviour. For more information on how the tag is generated, see `htmltools::tags()`.

Value

shiny tag object

See Also

Other Govstyle actions: `download_button()`, `download_link()`, `download_radios()`, `download_radios_handler()`

Examples

```
external_link("https://shiny.posit.co/", "R Shiny")

external_link(
  "https://shiny.posit.co/",
  "R Shiny",
  add_warning = FALSE
)

# Give sighted users a visual warning without repeating the text
external_link(
  "https://shiny.posit.co/",
  "R Shiny",
```

```
  add_warning = "icon"
)

# This will trim and show as 'R Shiny'
external_link("https://shiny.posit.co/", "  R Shiny")

# mailto: links skip the new-tab attributes, text, and icon
external_link("mailto:feedback@example.com", "feedback@example.com")

# Example of within text
shiny::tags$p(
  "Oi, ", external_link("https://shiny.posit.co/", "R Shiny"), " is great."
)

# Example of multiple links together
shiny::tags$h2("Related resources")
shiny::tags$p("The following links open in a new tab.")
gov_list(
  list = list(
    external_link(
      "https://shiny.posit.co/",
      "R Shiny documentation",
      add_warning = FALSE
    ),
    external_link(
      "https://www.python.org/",
      "Python documentation",
      add_warning = FALSE
    ),
    external_link(
      "https://nextjs.org/",
      "Next.js documentation",
      add_warning = FALSE
    )
  ),
  style = "bullet"
)
```

`file_Input` *File Input Function*

Description

This function create a file upload component. It uses the basis of the shiny fileInput function, but restyles the label and adds error onto it.

Usage

```
file_Input(
  inputId,
```

```
  label,
  multiple = FALSE,
  accept = NULL,
  width = NULL,
  buttonLabel = "Choose file",
  placeholder = "No file chosen",
  error = FALSE,
  error_message = NULL
)
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
label	Display label for the control, or NULL for no label
multiple	Whether the user should be allowed to select and upload multiple files at once. Does not work on older browsers, including Internet Explorer 9 and earlier
accept	A character vector of MIME types; gives the browser a hint of what kind of files the server is expecting
width	The width of the input, e.g. '400px', or '100\%'
buttonLabel	The label used on the button. Can be text or an HTML tag object
placeholder	The text to show before a file has been uploaded
error	If TRUE, render the component in its error state and reserve a slot for the error message. Defaults to FALSE.
error_message	Text shown when error is TRUE. Defaults to NULL.

Value

a file input HTML shiny tag object

See Also

Other Govstyle select inputs: button_Input(), checkbox_Input(), radio_button_Input(), select_Input(), update_radio_button_Input()

Examples

```
ui <- shinyGovstyle::gov_page(
  # Required for error handling function
  shinyjs::useShinyjs(),
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo="shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::phase_banner(
    inputId = "banner", type = "beta", 'This is a new service'
  ),
```

```
  shinyGovstyle::gov_main_layout(
    # Simple file input
    shinyGovstyle::file_Input(
      inputId = "file1",
      label = "Upload a file"
    ),
    # Error file
    shinyGovstyle::file_Input(
      inputId = "file2",
      label = "Upload a file",
      error = TRUE
    ),
    # Button to trigger error
    shinyGovstyle::button_Input(inputId = "submit", label = "Submit")
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {
  # Trigger error on blank submit of file2
  observeEvent(input$submit, {
    if (is.null(input$file2)){
      shinyGovstyle::error_on(inputId = "file2")
    } else {
      shinyGovstyle::error_off(
        inputId = "file2"
      )
    }
  })
}
if (interactive()) shinyApp(ui = ui, server = server)
```

font *Font Function*

Description

Loads the GDS Transport font for use in your app. GDS Transport is a restricted typeface that must only be used on GOV.UK domains. If your app is not hosted on a GOV.UK domain, do not call this function — the GOV.UK Frontend CSS will fall back to Helvetica or Arial automatically.

Usage

```
font()
```

Details

See the GOV.UK typeface guidance for full details on when GDS Transport is permitted.

Value

no value returned. This loads the font CSS file

Examples

```
ui <- shinyGovstyle::gov_page(
  font(),
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo="shinyGovstyle/images/moj_logo.png")
)

server <- function(input, output, session) {}

if (interactive()) shinyApp(ui = ui, server = server)
```

---

footer *Footer Function*

Description

This function create a gov style footer for your page

Usage

```
footer(full = FALSE, links = NULL, width = "standard")
```

Arguments

`full`	Whenever you want to have blank footer or official gov version. Defaults to `FALSE`. Not to be confused with the `width` argument below, which controls the footer's own width rather than its content.
`links`	A vector of actionLinks to be added to the footer, inputIDs are auto-generated and are the snake case version of the link text, e.g. "Accessibility Statement" will have an inputID of accessibility_statement
`width`	Width of the component. One of `"standard"` (the default, GOV.UK's usual 960px content width), `"three-quarters"` (three-quarters of the viewport, never narrower than standard), `"full"` (no max-width, so the container fills the viewport, with grid gutters also removed), or a CSS length (e.g. `"1400px"`, `"90vw"`) for a custom max-width.

Details

You can add links in the footer to content either within the dashboard or content external to the dashboard using the links_list argument.

Links in the footer should be used sparingly and are usually for supporting information pages such as the accessibility statement, privacy notice, cookies information or link to a statement of voluntary adoption of the statistics code of practice.

If adding a link to an internal page, generally you will be controlling a hidden tabset so to the end user it looks like it is a new page.

Value

a footer HTML shiny tag object

See Also

Other Govstyle page structure: cookieBanner(), gov_page(), header(), layouts, phase_banner(), skip_to_main()

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png",
    logo_alt_text = "Ministry of Justice logo"
  ),
  shinyGovstyle::phase_banner(
    inputId = "banner", type = "beta", "This is a new service"
  ),
  shinyGovstyle::gov_text("Placeholder text"),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {}

if (interactive()) shinyApp(ui = ui, server = server)

# Add links
footer(links = c("Accessibility statement", "Cookies"))

# Add links, internal and external
footer(
  links = c(
    "Accessibility statement",
    "Cookies",
    `Government Digital Service` =
    "https://www.gov.uk/government/organisations/government-digital-service"
  )
)
```

```
# Full app with link controlling a hidden tab and a link to an external page
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png",
    logo_alt_text = "Ministry of Justice logo"
  ),
  shinyGovstyle::phase_banner(
    inputId = "banner", type = "beta", "This is a new service"
  ),
  shiny::tabsetPanel(
    type = "hidden",
    id = "tabs",
    shiny::tabPanel(
      "Main content",
      value = "main",
      heading_text("Hello world!")
    ),
    shiny::tabPanel(
      "Accessibility statement",
      value = "accessibility-panel",
      heading_text("Accessibility statement")
    ),
    shiny::tabPanel(
      "Cookies",
      value = "cookies-panel",
      heading_text("Cookies")
    ),
  ),
  shinyGovstyle::footer(
    full = TRUE,
    links = c(
      `Accessibility statement` = "accessibility_footer_link",
      `Cookies` = "cookies_footer_link",
      `Government Digital Service` =
       paste0(
         "https://www.gov.uk/government/",
         "organisations/government-digital-service"
       )
    )
  )
)

server <- function(input, output, session) {
  shiny::observeEvent(input$accessibility_footer_link, {
    shiny::updateTabsetPanel(
      session,
      "tabs",
      selected = "accessibility-panel"
    )
  })
```

```
  shiny::observeEvent(input$cookies_footer_link, {
    shiny::updateTabsetPanel(
      session,
      "tabs",
      selected = "cookies-panel"
    )
  })
}

if (interactive()) shinyApp(ui = ui, server = server)
```

---

govReactable *Interactive govTable*

Description

This function is opinionated and sets table defaults that are in keeping with the wider GOV.UK design system. Some defaults can be overridden, such as `highlight=TRUE` and `borderless=TRUE`, however some are fixed, such as `showSortIcon=FALSE` as the default sort icon is inaccessible. Additional arguments from `reactable::reactable` can be passed to customise the table.

Usage

```
govReactable(
  df,
  right_col = NULL,
  page_size = 10,
  highlight = TRUE,
  borderless = TRUE,
  min_widths = list(),
  columns = list(),
  ...
)
```

Arguments

`df`	A dataframe used to generate the table
`right_col`	A vector of column names that should be right-aligned. By default, numeric data is right-aligned, and character data is left-aligned
`page_size`	The default number of rows displayed per page (default: 10)
`highlight`	Highlight table rows on hover
`borderless`	Remove inner borders from table
`min_widths`	Customise minimum column width using a list of columns and minimum width in pixels

`columns` Customise individual columns, for example to fix the number of decimal places shown. Give a named list, where each name matches a column in `df` and each value is built with `reactable::colDef()` (see examples). Any column left out of this list is unaffected, and keeps govReactable's usual GOV.UK look. For a column you do include, anything you don't set on it, such as sorting or alignment, also keeps that same default. Names that don't match a column in `df` are ignored.

`...` Additional arguments passed to `reactable::reactable`

Details

This function inserts a government-styled table using `reactable`. You can use this in R markdown or Quarto documents, or use renderGovReactable() and govReactableOutput() for tables in R Shiny. govReactableOutput() gives the ability to add a caption, for static tables made using just govReactable(), use heading_text() to add captions to tables.

Value

A `reactable` HTML widget styled with GOV.UK classes

See Also

Other Govstyle tables tabs and accordions: `accordion()`, `govReactable-shiny`, `govTable()`, `govTabs()`

Examples

```
# Example static table using govReactable
if (interactive()) {
  govReactable(
    iris,
    right_col = c(
      "Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width"
    )
  )

  govReactable(
    iris,
    right_col = c(
      "Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width"
    ),
    highlight = FALSE,
    page_size = 5,
    min_widths = list(
      Sepal.Length = 75,
      Sepal.Width = 75,
      Petal.Length = 75,
      Petal.Width = 75
    )
  )
```

```
  # Show one column to a fixed number of decimal places, leaving every
  # other column and style (sorting, alignment, etc.) untouched
  sales_data <- data.frame(
    shop = c("Shop A", "Shop B", "Shop C"),
    growth = c(4, 4.7, 12.34)
  )
  govReactable(
    sales_data,
    columns = list(
      growth = reactable::colDef(
        format = reactable::colFormat(digits = 1)
      )
    )
  )

  # A more involved example: only `percent` is customised, so `region`
  # and `count` keep the normal GOV.UK defaults untouched
  # (counts shown as whole numbers alongside a percentage column fixed
  # to 1 decimal place)
  count_pct_data <- data.frame(
    region = c("North", "South", "East"),
    count = c(1234, 56, 789),
    percent = c(4, 4.7, 12.34)
  )
  govReactable(
    count_pct_data,
    right_col = c("count", "percent"),
    columns = list(
      percent = reactable::colDef(
        format = reactable::colFormat(digits = 1)
      )
    )
  )
}
```

`govReactable-shiny` *Shiny bindings for govReactable Output and render functions for using govReactable within shiny apps*

Description

Shiny bindings for govReactable Output and render functions for using govReactable within shiny apps

Usage

```
govReactableOutput(
  output_table_name,
  caption,
  caption_size = "l",
```

```
  heading_level = 2L
)

renderGovReactable(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

`output_table_name`
: Output variable to read from

`caption`
: Adds a caption to the table as a header

`caption_size`
: Adjust the size of caption. One of `"s"`, `"m"`, `"l"`, `"xl"`, with `"l"` as the default. Any other value throws an error.

`heading_level`
: The HTML heading level for the caption, an integer from `2L` to `5L`. Defaults to `2L` (`<h2>`). Strings such as `"h3"` are **[Deprecated]** as of shinyGovstyle 0.3.0 and will stop working in shinyGovstyle 1.0.0; use the integer form (e.g. `3L`) instead.

`expr`
: An expression that generates a `reactable` widget

`env`
: The environment in which to evaluate `expr`

`quoted`
: Is `expr` a quoted expression (with `quote()`)? This is useful if you want to save an expression in a variable

Value

`govReactableOutput()` returns a `reactable` output element that can be included in a Shiny UI

`renderGovReactable()` returns a `reactable` render function that can be assigned to a Shiny output slot

See Also

Other Govstyle tables tabs and accordions: `accordion()`, `govReactable()`, `govTable()`, `govTabs()`

Examples

```
ui <- shinyGovstyle::gov_page(
  govReactableOutput(
    "table",
    caption = "Example table"
  )
)

server <- function(input, output, session) {
  output$table <- renderGovReactable({
    govReactable(iris)
  })
}

if (interactive()) shinyApp(ui, server)
```

govTable *Table Function*

Description

This function inserts a gov styled table. Format is with header looking rows and columns

Usage

```
govTable(
  inputId,
  df,
  caption,
  caption_size = "l",
  num_col = NULL,
  width_overwrite = NULL
)
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
df	expects a dataframe to create a table
caption	adds a caption to the table as a header
caption_size	adjust the size of caption. One of "s", "m", "l", "xl", with "l" as the default. Any other value throws an error.
num_col	adds numeric class format to these columns
width_overwrite	change width. Need to include width for every column. Columns must add up to 1. Options are three-quarters, two-thirds, one-half, one-third, one-quarter. Default is NULL

Details

For very large or interactive tables, consider govReactable() instead. govTable() renders a static HTML table, which becomes slow to render and harder to navigate as the number of rows grows. A warning is emitted when df has more than 50 rows.

Value

a table HTML shiny tag object

See Also

Other Govstyle tables tabs and accordions: accordion(), govReactable(), govReactable-shiny, govTabs()

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::gov_main_layout(
    shinyGovstyle::govTable(
      "tab1",
      shinyGovstyle::transport_data_small,
      "Test",
      "l",
      num_col = c(2,3),
      width_overwrite = c("one-half", "one-quarter", "one-quarter")
    )
  )
)

server <- function(input, output, session) {}

if (interactive()) shinyApp(ui = ui, server = server)
```

---

govTabs *Tabs Function*

Description

This function creates a tabs based table. It requires a single dataframe with a grouping variable.

Usage

```
govTabs(inputId, df, group_col)
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
df	A single dataframe with all data. See example for structure
group_col	The column name with the groups to be used as tabs

Value

a tab table HTML shiny tag object

See Also

Other Govstyle tables tabs and accordions: accordion(), govReactable(), govReactable-shiny, govTable()

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png",
    logo_alt_text = "Ministry of Justice logo"
  ),
  shinyGovstyle::gov_main_layout(
    shinyGovstyle::govTabs("tabs", shinyGovstyle::case_data, "tabs")
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {}
if (interactive()) shinyApp(ui = ui, server = server)
```

---

gov_list *Gov List function*

Description

Gov List function

Usage

```
gov_list(list, style = "none")
```

Arguments

list	vector or list of items. Each item accepts a plain character string, or shiny tag objects such as shiny::tags$a() (e.g. to add a link to a bullet) or a shiny::tagList().
style	options: "none", "bullet", "number". defaults to "none"

Value

A Shiny tag representing an ordered or unordered HTML list.

See Also

Other Govstyle text types: date_Input(), heading_text(), input_field(), text_Input(), text_area_Input()

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples"
  ),
  shinyGovstyle::phase_banner(
    inputId = "banner", type = "beta", 'This is a new service'
  ),
  shinyGovstyle::gov_main_layout(
    shinyGovstyle::heading_text("gov_list", size = "s"),
    shinyGovstyle::gov_text("List:"),
    gov_list(list = c("a", "b", "c")),
    shinyGovstyle::gov_text("Bulleted list:"),
    gov_list(list = c("a", "b", "c"), style = "bullet"),
    shinyGovstyle::gov_text("Numbered list:"),
    gov_list(list = c("one", "two", "three"), style = "number"),
    shinyGovstyle::gov_text("List with a link:"),
    gov_list(
      list = list(
        "Plain item",
        shiny::tags$a(href = "https://www.gov.uk", "A link"),
        shiny::tagList("Item with ", shiny::tags$b("bold"), " text")
      ),
      style = "bullet"
    )
  )
)

server <- function(input, output, session) {}

if (interactive()) shinyApp(ui = ui, server = server)
```

---

gov_page *GOV.UK styled page wrapper*

Description

`gov_page()` wraps `bslib::page_fluid()` so a shinyGovstyle app starts from GOV.UK-friendly defaults instead of a blank page. `bslib` is now generally the recommended way to build R Shiny UIs (rather than `shiny::fluidPage()`), so shinyGovstyle builds on it too; you can mix other `bslib` components into a shinyGovstyle app freely. `gov_page()` sets the page language (important for screen readers), lets you add a page description, and lets you set a default width for every shinyGovstyle component in the page in one place, instead of repeating `width =` on each of them.

Usage

```
gov_page(
  ...,
```

```
  title = NULL,
  lang = "en",
  description = NULL,
  width = "full",
  theme = bslib::bs_theme()
)
```

Arguments

...	The rest of your page: header(), phase_banner(), service_navigation(), gov_main_layout(), footer(), and so on.
title	Browser tab title, passed straight to bslib::page_fluid().
lang	The page's language, as an ISO language code # nolint (e.g. "en", "cy" for Welsh). Screen readers use this to choose the right pronunciation and voice, so it should always be set correctly. Defaults to "en".
description	Short summary of the page, added as a <meta name="description"> tag. Used by search engines and some assistive technology. Defaults to NULL (no description tag). Keep it to one concise sentence: search engines typically truncate meta descriptions at around 150-160 characters.
width	Default width for every shinyGovstyle component used inside ... that doesn't set its own width. One of "full" (the default, no max-width, so the page fills the viewport, with grid gutters also removed), "standard" (GOV.UK's usual 960px content width, the better choice for an ordinary content-style page), "three-quarters" (three-quarters of the viewport, never narrower than standard), or a CSS length (e.g. "1400px", "90vw") for a custom max-width. A component that sets its own width always overrides this default.
theme	A bslib::bs_theme() object, passed straight to bslib::page_fluid().

Value

a page HTML shiny tag object

See Also

Other Govstyle page structure: cookieBanner(), footer(), header(), layouts, phase_banner(), skip_to_main()

Examples

```
ui <- shinyGovstyle::gov_page(
  title = "My dashboard",
  description = "A dashboard showing my department's latest statistics",
  width = "three-quarters",
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "My dashboard"
  ),
  shinyGovstyle::gov_main_layout(
    shinyGovstyle::heading_text("Welcome", size = "l", level = 1)
```

```
  ),
  shinyGovstyle::footer()
)

server <- function(input, output, session) {}

if (interactive()) shinyApp(ui = ui, server = server)
```

---

gov_summary *Tabs Function*

Description

This function creates a tabs based table. It requires a single dataframe with a grouping variable.

Usage

```
gov_summary(inputId, headers, info, action = FALSE, border = TRUE)
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
headers	input for the row headers value
info	Summary information values for the table. Each value accepts a plain character string, or shiny tag objects such as shiny::tags$b("Bold") or a shiny::tagList(). Pass a list rather than a character vector to mix tag and string values.
action	whenever a change link is needed. Sets input to the value of the headers using lowercase and with underscore to replace gaps. Default set to FALSE
border	set if the table should have borders. Default set to TRUE

Value

a summary list table HTML shiny tag object

See Also

Other Govstyle feedback types: confirmation_panel(), details(), gov_tag(), inset_text(), label_hint(), noti_banner(), value_box(), warning_text()

Examples

```
# Create an example dataset
headers <- c(
  "Name",
  "Date of birth",
  "Contact information",
  "Contact details"
```

```
)
info <- c(
  "Sarah Philips",
  "5 January 1978",
  "72 Guild Street <br> London <br> SE23 6FH",
  "07700 900457 <br> sarah.phillips@example.com"
)

ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo="shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::gov_main_layout(
    shinyGovstyle::gov_summary("sumID", headers, info, action = FALSE)
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {}
if (interactive()) shinyApp(ui = ui, server = server)
```

---

gov_tag *Tag*

Description

This function creates the GOV.UK tag component, used to show users the status of something.

Usage

```
gov_tag(inputId, text, colour = "navy")
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
text	The text in the tag
colour	The colour of the tag. Default is navy. Other options are grey, green, teal, blue, purple, magenta, red, orange and yellow. Any other value gives a warning listing these options

Value

a tag HTML shiny tag object

See Also

Other Govstyle feedback types: confirmation_panel(), details(), gov_summary(), inset_text(), label_hint(), noti_banner(), value_box(), warning_text()

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::gov_main_layout(
    shinyGovstyle::gov_tag("tag1", "Complete"),
    shinyGovstyle::gov_tag("tag2", "Incomplete", "red")
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {}
if (interactive()) shinyApp(ui = ui, server = server)
```

---

header *Header Function*

Description

This function create a header banner. For use at top of the screen

Usage

```
header(
  org_name = "Shiny example app",
  service_name = NULL,
  logo = "shinyGovstyle/images/Dept_logo.svg",
  main_text = lifecycle::deprecated(),
  secondary_text = lifecycle::deprecated(),
  main_link = lifecycle::deprecated(),
  secondary_link = lifecycle::deprecated(),
  logo_alt_text = "Departmental logo",
  main_alt_text = lifecycle::deprecated(),
  secondary_alt_text = lifecycle::deprecated(),
  logo_width = 66,
  logo_height = 34,
  width = "standard"
)
```

Arguments

`org_name`	Organisation name that goes in the header
`service_name`	Service name to supplement the organisation name
`logo`	Add a link to a logo which will apply in the header. Use crown to use the crown SVG version on GOV UK
`main_text`	**[Deprecated]** Use `org_name` instead
`secondary_text`	**[Deprecated]** Use `service_name` instead
`main_link`	**[Deprecated]** Has no effect and will be removed in shinyGovstyle 1.0.0
`secondary_link`	**[Deprecated]** Has no effect and will be removed in shinyGovstyle 1.0.0
`logo_alt_text`	Add alternative text for the logo. Should be used when a logo is used
`main_alt_text`	**[Deprecated]** Has no effect and will be removed in shinyGovstyle 1.0.0
`secondary_alt_text`	**[Deprecated]** Has no effect and will be removed in shinyGovstyle 1.0.0
`logo_width`	Change the logo size width CSS to improve fit
`logo_height`	Change the logo size height CSS to improve fit
`width`	Width of the component. One of `"standard"` (the default, GOV.UK's usual 960px content width), `"three-quarters"` (three-quarters of the viewport, never narrower than standard), `"full"` (no max-width, so the container fills the viewport, with grid gutters also removed), or a CSS length (e.g. `"1400px"`, `"90vw"`) for a custom max-width.

Value

a header HTML shiny tag object

See Also

Other Govstyle page structure: `cookieBanner()`, `footer()`, `gov_page()`, `layouts`, `phase_banner()`, `skip_to_main()`

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png",
    logo_alt_text = "Ministry of Justice Logo"
  )
)

server <- function(input, output, session) {}

if (interactive()) shinyApp(ui = ui, server = server)
```

heading_text *Heading Text Function*

Description

This function creates heading text

Usage

```
heading_text(text_input, size = "xl", id, level = 1)
```

Arguments

text_input	Text to display
size	Text size, one of "xl", "l", "m", "s". Defaults to "xl". Any other value throws an error.
id	Custom header id
level	Heading level, integer between 1 and 6. Defaults to 1

Value

a heading text HTML shiny tag object

See Also

Other Govstyle text types: date_Input(), gov_list(), input_field(), text_Input(), text_area_Input()

Examples

```
shinyGovstyle::heading_text("This is great text")
shinyGovstyle::heading_text("This is great text", size = "l", level = 2)
```

input_field *Input Field Function*

Description

This function inserts number of text inputs. Useful for addresses.

Usage

```
input_field(
  legend,
  labels,
  inputIds,
  widths = NULL,
  types = "text",
  error = FALSE,
  error_message = NULL
)
```

Arguments

legend	Legend that goes above the fieldset
labels	A list of labels for the text inputs
inputIds	A list input slots that will be used to access the value
widths	control the size of the box based on number of characters required. Options are 30, 20, 10, 5, 4, 3, 2. NULL will not limit the size
types	text box types. Will default to text
error	If TRUE, render the component in its error state and reserve a slot for the error message. Defaults to FALSE.
error_message	Text shown when error is TRUE. Defaults to NULL.

Value

a input field of HTML as a shiny tag object

See Also

Other Govstyle text types: date_Input(), gov_list(), heading_text(), text_Input(), text_area_Input()

Examples

```
ui <- shinyGovstyle::gov_page(
  # Required for error handling function
  shinyjs::useShinyjs(),
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo="shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::phase_banner(
    inputId = "banner", type = "beta", 'This is a new service'
  ),
  shinyGovstyle::gov_main_layout(
    shinyGovstyle::input_field(
      legend ="List of three text boxes in a field",
      labels = c("Field 1", "Field 2", "Field 3"),
```

```
      inputIds = c("field1", "field2", "field3"),
      widths = c(30,20,10),
      error = TRUE
    ),
    # Button to trigger error
    shinyGovstyle::button_Input(inputId = "submit", label = "Submit")
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {
  # Trigger error on blank submit of field2
  observeEvent(input$submit, {
    if (input$field2 == ""){
      shinyGovstyle::error_on(
        inputId = "field2",
        error_message = "Please complete"
      )
    } else {
      shinyGovstyle::error_off(
        inputId = "field2"
      )
    }
  })
}
if (interactive()) shinyApp(ui = ui, server = server)
```

`inset_text` *Inset text*

Description

This function creates the GOV.UK inset text component, used to differentiate a block of text from the content that surrounds it, for example a quote, example, or additional information about the page.

Usage

```
inset_text(inputId, content)
```

Arguments

`inputId`	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via `input$<inputId>`.
`content`	Content to display in the inset. Accepts a plain character string, or `shiny` tag objects such as `shiny::tags$b("Bold")` or a `shiny::tagList()`.

Value

an inset text HTML shiny tag object

See Also

Other Govstyle feedback types: confirmation_panel(), details(), gov_summary(), gov_tag(), label_hint(), noti_banner(), value_box(), warning_text()

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::gov_main_layout(
    shinyGovstyle::inset_text(
      inputId = "note",
      content = paste(
        "It can take up to 8 weeks to register a lasting power of",
        "attorney if there are no mistakes in the application."
      )
    ),
    shinyGovstyle::inset_text(
      inputId = "note-rich",
      content = shiny::tagList(
        shiny::tags$b("Important: "),
        "you can also pass tag objects."
      )
    )
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {}
if (interactive()) shinyApp(ui = ui, server = server)
```

---

label_hint *Label with Hint Function*

Description

This function inserts a label and optional hint.

Usage

```
label_hint(inputId, label, hint_input = NULL)
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.

label	Display label for the control, or NULL for no label. Accepts a plain character string, an HTML string, or shiny tag objects such as shiny::tags$b("Bold") or a shiny::tagList().
hint_input	Display hint label for the control, or NULL for no hint label. Accepts the same rich content as label, so it can include a link.

Value

a label hint HTML shiny tag object

See Also

Other Govstyle feedback types: confirmation_panel(), details(), gov_summary(), gov_tag(), inset_text(), noti_banner(), value_box(), warning_text()

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo="shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::gov_main_layout(
    label_hint(
      inputId = "label1",
      label = "This is a label",
      hint_input = "This is a hint"
    ),
    # Rich content: a link in the hint
    label_hint(
      inputId = "label2",
      label = "Bold label",
      hint_input = shiny::tagList(
        "See the ",
        shinyGovstyle::external_link(
          "https://www.gov.uk",
          "GOV.UK guidance"
        )
      )
    )
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {}
if (interactive()) shinyApp(ui = ui, server = server)
```

layouts *Page Layout Functions*

Description

Build the main content area of a GOV.UK page. `gov_main_layout()` creates the page's `<main>` landmark, and content can go straight inside it: items simply stack one below the other. Use `gov_row()` and `gov_box()` only when you need GOV.UK grid columns, such as two boxes side by side or a section with a set column width. `gov_text()` is a container for body text.

Usage

```
gov_main_layout(..., inputID = "main", width = "standard")

gov_row(...)

gov_box(..., size = "full")

gov_text(...)
```

Arguments

`...`	include the components of the UI that you want within the main page. These components are made to flow through each other. See example
`inputID`	ID of the main div. Defaults to "main"
`width`	Width of the component. One of `"standard"` (the default, GOV.UK's usual 960px content width), `"three-quarters"` (three-quarters of the viewport, never narrower than standard), `"full"` (no max-width, so the container fills the viewport, with grid gutters also removed), or a CSS length (e.g. `"1400px"`, `"90vw"`) for a custom max-width.
`size`	size of the box in the row. Optional are full, one-half, two-thirds, one-third, three-quarters and one-quarter. Defaults to "full"

Value

a HTML shiny layout div

See Also

Other Govstyle page structure: `cookieBanner()`, `footer()`, `gov_page()`, `header()`, `phase_banner()`, `skip_to_main()`

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo="shinyGovstyle/images/moj_logo.png"),
  shinyGovstyle::gov_main_layout(
    # Content with no columns goes straight into gov_main_layout()
    shinyGovstyle::heading_text("Page heading", size = "l"),
    shinyGovstyle::gov_text("Items stack one below the other."),
    # Use gov_row() and gov_box() for side-by-side columns
    shinyGovstyle::gov_row(
      shinyGovstyle::gov_box(
        size = "one-half",
        shinyGovstyle::gov_text("govuk-grid-column-one-half")
      ),
      shinyGovstyle::gov_box(
        size = "one-half",
        shinyGovstyle::gov_text("govuk-grid-column-one-half")
      )
    ),
    shinyGovstyle::gov_row(
      shinyGovstyle::gov_box(
        size = "one-third",
        shinyGovstyle::gov_text("govuk-grid-column-one-third")
      ),
      shinyGovstyle::gov_box(
        size = "two-thirds",
        shinyGovstyle::gov_text("govuk-grid-column-two-thirds")
      )
    ),
    shinyGovstyle::gov_row(
      shinyGovstyle::gov_box(
        size = "one-quarter",
        shinyGovstyle::gov_text("govuk-grid-column-one-quarter")
      ),
      shinyGovstyle::gov_box(
        size = "three-quarters",
        shinyGovstyle::gov_text("govuk-grid-column-three-quarters")
      )
    )
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {}

if (interactive()) shinyApp(ui = ui, server = server)
```

---

navigate_to *Navigate to a page in one call*

Description

Switches the visible tab panel and updates the active service navigation link in a single call. Use this for programmatic navigation — next / back buttons, modal links, footer shortcuts to a main page — where the user has not clicked a nav link directly.

When `service_navigation()` is used with `auto_page_title = TRUE` (the default), the browser tab title is updated automatically too, because `update_service_navigation()` triggers the title sync via the JavaScript binding.

By default the nav link `inputId` is also used as the target tab panel `value`. When they differ — for example when nav inputIds carry a prefix like `sn_` to disambiguate from panel values — pass `panel` explicitly.

Usage

```
navigate_to(session, tabset_id, inputId, panel = inputId)
```

Arguments

`session`	The Shiny session object
`tabset_id`	The `id` of the `shiny::tabsetPanel()` or `bslib::navset_hidden()` to switch
`inputId`	The nav link inputId to set as the active item in the service navigation. It is cleaned up the same way as in `service_navigation()`, so `"Page-Two"` and `"page_two"` both refer to the same link
`panel`	The `value` of the tab panel to switch to. Defaults to `inputId` for the common case where the two are the same. When your nav inputIds and panel values differ (e.g. nav id `sn_cookies`, panel value `panel-cookies`), pass `panel` explicitly. If you omit it when the values differ, only the nav highlight moves, the visible tab stays put.

Value

NULL, called for side effects

See Also

Other Govstyle navigation: `backlink_Input()`, `service_navigation()`, `service_navigation_server()`, `update_page_title()`, `update_service_navigation()`

Examples

```
# Next / back buttons that cross page boundaries - inputId and panel
# value match, so a single id is enough.
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::service_navigation(c("Page one", "Page two")),
  bslib::navset_hidden(
    id = "tabs",
    bslib::nav_panel("page_one", shiny::actionButton("next_btn", "Next")),
    bslib::nav_panel("page_two", "Page two content")
  )
```

```
)

server <- function(input, output, session) {
  shiny::observeEvent(input$next_btn, {
    shinyGovstyle::navigate_to(session, "tabs", "page_two")
  })
}

if (interactive()) shiny::shinyApp(ui = ui, server = server)

# Mismatched inputId and panel value — name the panel explicitly.
# The nav links use an `sn_` prefix on their inputIds so they don't clash
# with any other inputs in the app. The panel values are unrelated names.
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::service_navigation(
    # names = link text shown to the user, values = nav inputIds
    c(Home = "sn_home", Cookies = "sn_cookies")
  ),
  bslib::navset_hidden(
    id = "tabs",
    # first arg of nav_panel() is the panel value
    bslib::nav_panel("start", shiny::actionButton("cookies_btn", "Cookies")),
    bslib::nav_panel("policy", "Cookies content")
  )
)

server <- function(input, output, session) {
  shiny::observeEvent(input$cookies_btn, {
    # inputId = nav link to highlight; panel = tab panel to show
    shinyGovstyle::navigate_to(
      session, "tabs", inputId = "sn_cookies", panel = "policy"
    )
  })
}

if (interactive()) shiny::shinyApp(ui = ui, server = server)
```

---

noti_banner *Notification Banner Function*

Description

This function creates a notification banner.

Usage

```
noti_banner(
  inputId,
  title_txt = "Important",
  body_txt = NULL,
```

```
  type = "standard"
)
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
title_txt	The wording that appears in the title
body_txt	The wording that appears in the banner body. Accepts a plain character string, or shiny tag objects such as shiny::tags$b("Bold") or a shiny::tagList().
type	The type of banner. Options are standard and success. Standard is default

Value

a notification HTML shiny tag object

See Also

Other Govstyle feedback types: confirmation_panel(), details(), gov_summary(), gov_tag(), inset_text(), label_hint(), value_box(), warning_text()

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo="shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::noti_banner(
    inputId = "banner", title_txt = "Important", body_txt = "Example text"
  )
)

server <- function(input, output, session) {}

if (interactive()) shinyApp(ui = ui, server = server)
```

---

phase_banner *Phase banner*

Description

This function creates the GOV.UK phase banner component, used to show that a service is still being worked on, for example in alpha or beta.

Usage

```
phase_banner(
  inputId,
  type,
  label = NULL,
  feedback_url = NULL,
  width = "standard"
)
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
type	Main type of label e.g. alpha or beta. Can be any word
label	Text to display. Accepts a plain character string, or shiny tag objects such as shiny::tags$b("Bold") or a shiny::tagList(). Not required if feedback_url is supplied instead.
feedback_url	Optional URL used to auto-generate the standard GOV.UK phase banner feedback text, e.g. "This is a new service - your feedback (opens in new tab) will help us to improve it.", with feedback linking to feedback_url. If feedback_url starts with mailto: the text instead reads "This is a new service - please contact [email address] if you have any questions or feedback.". Exactly one of label or feedback_url must be supplied.
width	Width of the component. One of "standard" (the default, GOV.UK's usual 960px content width), "three-quarters" (three-quarters of the viewport, never narrower than standard), "full" (no max-width, so the container fills the viewport, with grid gutters also removed), or a CSS length (e.g. "1400px", "90vw") for a custom max-width.

Value

a phase banner HTML shiny tag object

See Also

Other Govstyle page structure: cookieBanner(), footer(), gov_page(), header(), layouts, skip_to_main()

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::phase_banner(
    inputId = "banner", type = "Beta", 'This is a new service'
```

```
  )
)

server <- function(input, output, session) {}

if (interactive()) shinyApp(ui = ui, server = server)

# Auto-generate the standard feedback text from a URL
shinyGovstyle::phase_banner(
  inputId = "banner",
  type = "Beta",
  feedback_url = "https://example.com/feedback"
)

# Auto-generate contact text from a mailto: link
shinyGovstyle::phase_banner(
  inputId = "banner",
  type = "Beta",
  feedback_url = "mailto:feedback@example.com"
)
```

---

radio_button_Input *Radio Button Function*

Description

This function create radio buttons

Usage

```
radio_button_Input(
  inputId,
  label,
  choices = NULL,
  selected = NULL,
  inline = FALSE,
  small = FALSE,
  choiceNames = NULL,
  choiceValues = NULL,
  hint_label = NULL,
  error = FALSE,
  error_message = NULL,
  custom_class = "",
  label_size = c("m", "s", "l", "xl"),
  heading_level = NULL
)
```

Arguments

- `inputId` — The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via `input$<inputId>`.
- `label` — Display label for the control, or `NULL` for no label. Accepts a plain character string, an HTML string, or `shiny` tag objects such as `shiny::tags$b("Bold")` or a `shiny::tagList()`.
- `choices` — List of values to select from (if elements of the list are named then that name rather than the value is displayed to the user)
- `selected` — The initially selected value.
- `inline` — If you want the radio inline or not, Default is FALSE
- `small` — If you want the smaller versions of radio buttons, Default is FALSE
- `choiceNames, choiceValues` — Same as in `shiny::checkboxGroupInput()`. List of names and values, respectively, that are displayed to the user in the app and correspond to the each choice (for this reason they must have the same length). If either of these arguments is provided, then the other must be provided and choices must not be provided. The advantage of using both of these over a named list for choices is that choiceNames allows any type of UI object to be passed through (tag objects, icons, HTML code, ...), instead of just simple text
- `hint_label` — Display hint label for the control, or `NULL` for no hint label. Accepts the same rich content as `label`, so it can include a link.
- `error` — If `TRUE`, render the component in its error state and reserve a slot for the error message. Defaults to `FALSE`.
- `error_message` — Text shown when `error` is `TRUE`. Defaults to `NULL`.
- `custom_class` — If you want to add additional classes to the radio buttons
- `label_size` — Size modifier for the legend. One of `"m"`, `"s"`, `"l"`, or `"xl"`, matching the GDS `govuk-fieldset__legend--*` classes. Defaults to `"m"`.
- `heading_level` — Optional heading level for the legend. If supplied (an integer 1-6), the legend text is wrapped in a `<hN>` with the GDS `govuk-fieldset__heading` class, following the GDS pattern for using a question as the page heading. Defaults to `NULL` (no heading wrap).

Value

radio buttons HTML shiny tag object

See Also

Other Govstyle select inputs: `button_Input()`, `checkbox_Input()`, `file_Input()`, `select_Input()`, `update_radio_button_Input()`

Examples

```
ui <- shinyGovstyle::gov_page(
  # Required for error handling function
  shinyjs::useShinyjs(),
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::phase_banner(
    inputId = "banner", type = "beta", "This is a new service"
  ),
  shinyGovstyle::gov_main_layout(
    # Simple radio
    shinyGovstyle::radio_button_Input(
      inputId = "radio1",
      choices = c("Yes", "No", "Maybe"),
      label = "Choice option"
    ),
    # Error radio
    shinyGovstyle::radio_button_Input(
      inputId = "radio2",
      choices = c("Yes", "No", "Maybe"),
      label = "Choice option",
      hint_label = "Select the best fit",
      inline = TRUE,
      error = TRUE,
      error_message = "Select one"
    ),
    # Rich content: a link in the hint
    shinyGovstyle::radio_button_Input(
      inputId = "radio3",
      choices = c("Yes", "No", "Maybe"),
      label = "Choice option",
      hint_label = shiny::tagList(
        "See the ",
        shinyGovstyle::external_link(
          "https://www.gov.uk",
          "GOV.UK guidance"
        )
      )
    ),
    # Button to trigger error
    shinyGovstyle::button_Input(inputId = "submit", label = "Submit")
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {
  # Trigger error on blank submit of eventId2
  observeEvent(input$submit, {
    if (is.null(input$radio2)) {
```

```
      shinyGovstyle::error_on(inputId = "radio2")
    } else {
      shinyGovstyle::error_off(
        inputId = "radio2"
      )
    }
  })
}
if (interactive()) shinyApp(ui = ui, server = server)
```

`run_example` *Run examples*

Description

This function runs an example R Shiny app showcasing different parts of the package. Code for the app can be found in the inst/example_app folder in the source code.

Usage

```
run_example()
```

Details

The app uses the bslib package as is generally recommended for Shiny apps.

The app is also deployed at the following URL: https://department-for-education.shinyapps.io/shinygovstyle-example-app/

Value

runs an R Shiny app with examples in

Examples

```
if (interactive()) run_example()
```

`select_Input` *Select Function*

Description

This function inserts a select box

Usage

```
select_Input(inputId, label, select_text, select_value)
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
label	Display label for the control, or NULL for no label. Accepts a plain character string, an HTML string, or shiny tag objects such as shiny::tags$b("Bold") or a shiny::tagList().
select_text	Add the text that will apply in the drop down as a list
select_value	Add the value that will be used for each selection

Value

a select input HTML shiny tag object

See Also

Other Govstyle select inputs: button_Input(), checkbox_Input(), file_Input(), radio_button_Input(), update_radio_button_Input()

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png",
    logo_alt_text = "Ministry of Justice logo"
  ),
  shinyGovstyle::gov_main_layout(
    select_Input(
      inputId = "sorter",
      label = "Sort by",
      select_text = c(
        "Recently published",
        "Recently updated",
        "Most views",
        "Most comments"
      ),
      select_value = c("published", "updated", "view", "comments")
    ),
    shinyGovstyle::gov_text("Placeholder text")
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {}
if (interactive()) shinyApp(ui = ui, server = server)
```

service_navigation *Service navigation*

Description

Service navigation component consistent with the GDS service navigation. # nolint

Note: This component uses a hardcoded element ID for the navigation list and mobile menu toggle. Using multiple `service_navigation()` instances on the same page will cause ID conflicts and the mobile menu toggle may not work correctly.

The first link is marked as the current page when the app loads, matching the first tab panel Shiny shows by default. If your app starts on a different page, call `update_service_navigation()` when the session starts.

Put `service_navigation()` in your app's top-level UI and wire it up from the top-level server function. It is not designed to be used inside a Shiny module: an inputID wrapped in `ns()` is cleaned up like any other (so `NS("mod", "summary")` becomes `mod_summary`), and no longer matches the module's namespaced input.

Usage

```
service_navigation(
  links,
  service_name = NULL,
  auto_page_title = TRUE,
  page_title_suffix = NULL,
  width = "standard"
)
```

Arguments

`links` A character vector of link text for the service navigation. Each link's inputID is generated by lowercasing the text and replacing every non-alphanumeric character with an underscore, e.g. "Overview page" becomes `overview_page`. To choose the inputIDs yourself, supply a named vector where names are the link text and values are the inputIDs, e.g. `c("Overview page" = "overview")`. Supplied inputIDs are cleaned up in the same way, so `"Page-Two"` becomes `page_two`. Every link must end up with a unique inputID.

`service_name` An optional character string containing the service name to be displayed in the navigation bar

`auto_page_title` Logical. When `TRUE` (the default), the browser tab title is updated to match the active nav link whenever the user navigates — both by clicking a link and via `update_service_navigation()`. Set to `FALSE` to opt out and manage the page title yourself. Keeping the title in sync with the visible page is a GOV.UK Design System recommendation for accessibility.

page_title_suffix
: Optional character string appended to the page title in the format "<link text> | <suffix>" — typically the service name. Only used when auto_page_title = TRUE. For finer control (e.g. when a page heading differs from its nav link label), call update_page_title() from your server code.

width
: Width of the component. One of "standard" (the default, GOV.UK's usual 960px content width), "three-quarters" (three-quarters of the viewport, never narrower than standard), "full" (no max-width, so the container fills the viewport, with grid gutters also removed), or a CSS length (e.g. "1400px", "90vw") for a custom max-width.

Value

Shiny tag object

See Also

Other Govstyle navigation: backlink_Input(), navigate_to(), service_navigation_server(), update_page_title(), update_service_navigation()

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::header("Title", "Secondary heading"),
  shinyGovstyle::service_navigation(
     c("Summary data", "Detailed stats 1", "User guide")
  ),
  bslib::navset_hidden(
     id = "main_panels",
     bslib::nav_panel(
       "summary_data",
       shiny::tags$h2("Summary data")
     ),
     bslib::nav_panel(
       "detailed_stats_1",
       shiny::tags$h2("Detailed stats 1")
     ),
     bslib::nav_panel(
       "user_guide",
       shiny::tags$h2("User guide")
     )
   ),
   shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {
  observeEvent(
    input$summary_data,
    bslib::nav_select("main_panels", "summary_data")
  )
  observeEvent(
```

```
    input$detailed_stats_1,
    bslib::nav_select("main_panels", "detailed_stats_1")
  )
  observeEvent(
    input$user_guide,
    bslib::nav_select("main_panels", "user_guide")
  )
}

if (interactive()) shiny::shinyApp(ui = ui, server = server)
```

`service_navigation_server`

Auto-wire service navigation links to tabset panels

Description

Sets up one observer per nav link that switches the corresponding tab panel when the link is clicked. Eliminates the per-link `observeEvent()` boilerplate that multi-page apps would otherwise need.

Call once in your server function, after `service_navigation()` has been added to the UI. The active nav highlight and (when `auto_page_title = TRUE`) the browser tab title are already handled client-side, so no additional wiring is needed.

Usage

```
service_navigation_server(session, tabset_id, link_to_panel)
```

Arguments

`session`	The Shiny session object
`tabset_id`	The `id` of the `shiny::tabsetPanel()` or `bslib::navset_hidden()` to switch
`link_to_panel`	A character vector mapping nav link inputIds to tab panel values. Pass a **named** vector when inputIds differ from panel values (names = inputIds, values = panel values). Pass an **unnamed** vector as shorthand for the 1:1 case — each element is used as both the nav inputId and the panel value. InputIds are cleaned up the same way as in `service_navigation()`, so `"Page-Two"` and `"page_two"` both refer to the same link. Panel values are used exactly as given.

Value

NULL, called for side effects

See Also

Other Govstyle navigation: `backlink_Input()`, `navigate_to()`, `service_navigation()`, `update_page_title()`, `update_service_navigation()`

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::service_navigation(
    c(Summary = "sn_summary", Detail = "sn_detail")
  ),
  bslib::navset_hidden(
    id = "tabs",
    bslib::nav_panel("summary", "Summary content"),
    bslib::nav_panel("detailed_data", "Detail content")
  )
)

server <- function(input, output, session) {
  # Nav inputIds differ from panel values
  shinyGovstyle::service_navigation_server(
    session,
    tabset_id = "tabs",
    link_to_panel = c(
      sn_summary = "summary",
      sn_detail = "detailed_data"
    )
  )

  # Or, when inputIds match panel values exactly:
  # shinyGovstyle::service_navigation_server(
  #   session,
  #   tabset_id = "tabs",
  #   link_to_panel = c("summary", "detailed_data")
  # )
}

if (interactive()) shiny::shinyApp(ui = ui, server = server)
```

`skip_to_main` *Skip to main content link*

Description

This function generates a 'Skip to main content' link, which is typically used by keyboard users to bypass content and navigate directly to the main content of a page.

Usage

```
skip_to_main(id = "main")
```

Arguments

`id`	An optional parameter to specify the Id of the main content section, will be automatically preceded by a hash '#'. Default is "main" to match the "#main" Id within `gov_main()`.

Value

A Shiny tag representing the 'Skip to main content' link

See Also

Other Govstyle page structure: cookieBanner(), footer(), gov_page(), header(), layouts, phase_banner()

Examples

```
ui <- shinyGovstyle::gov_page(
  skip_to_main(),
  header(
    org_name = "Example",
    service_name = "User Examples",
    logo="shinyGovstyle/images/moj_logo.png"
  ),
  gov_main_layout(
    heading_text("Example heading"),
  )
)

server <- function(input, output, session){}

if (interactive()) shinyApp(ui = ui, server = server)
```

---

text_area_Input *Text Area Input Function*

Description

This function create a text area input.

Usage

```
text_area_Input(
  inputId,
  label,
  hint_label = NULL,
  row_no = 5,
  error = FALSE,
  error_message = NULL,
  word_limit = NULL
)
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
label	Display label for the control, or NULL for no label. Accepts a plain character string, an HTML string, or shiny tag objects such as shiny::tags$b("Bold") or a shiny::tagList().
hint_label	Display hint label for the control, or NULL for no hint label. Accepts the same rich content as label, so it can include a link.
row_no	Size of the text entry box. Defaults to 5
error	If TRUE, render the component in its error state and reserve a slot for the error message. Defaults to FALSE.
error_message	Text shown when error is TRUE. Defaults to NULL.
word_limit	Add a word limit to the display. Defaults to NULL

Value

a text area box HTML shiny tag object

See Also

Other Govstyle text types: date_Input(), gov_list(), heading_text(), input_field(), text_Input()

Examples

```
text_area_Input(
  "taId",
  "Can you provide more detail?",
  paste(
    "Do not include personal or financial information, like your",
    "National Insurance number or credit card details."
  )
)

# Rich content: a link in the hint
text_area_Input(
  "taId2",
  "Can you provide more detail?",
  shiny::tagList(
    "Read the ",
    shinyGovstyle::external_link("https://www.gov.uk", "guidance on detail")
  )
)
```

text_Input — *Text Input Function*

Description

This function create a text input.

Usage

```
text_Input(
  inputId,
  label,
  hint_label = NULL,
  type = "text",
  width = NULL,
  error = FALSE,
  error_message = NULL,
  prefix = NULL,
  suffix = NULL
)
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
label	Display label for the control, or NULL for no label. Accepts a plain character string, an HTML string, or shiny tag objects such as shiny::tags$b("Bold") or a shiny::tagList().
hint_label	Display hint label for the control, or NULL for no hint label. Accepts the same rich content as label, so it can include a link.
type	Type of text input to accept. Defaults to text
width	control the size of the box based on number of characters required. Options are 30, 20, 10, 5, 4, 3, 2. NULL will not limit the size
error	If TRUE, render the component in its error state and reserve a slot for the error message. Defaults to FALSE.
error_message	Text shown when error is TRUE. Defaults to NULL.
prefix	Add a prefix to the box. Defaults to NULL
suffix	Add a suffix to the box. Defaults to NULL

Value

a text input HTML shiny tag object

See Also

Other Govstyle text types: date_Input(), gov_list(), heading_text(), input_field(), text_area_Input()

Examples

```
ui <- shinyGovstyle::gov_page(
  # Required for error handling function
  shinyjs::useShinyjs(),
  shinyGovstyle::header(
    org_name = "Example",
    service_name = "User Examples",
    logo = "shinyGovstyle/images/moj_logo.png"
  ),
  shinyGovstyle::phase_banner(
    inputId = "banner", type = "beta", 'This is a new service'
  ),
  shinyGovstyle::gov_main_layout(
    # Simple text box
    shinyGovstyle::text_Input(inputId = "eventId", label = "Event Name"),
    # Error text box
    shinyGovstyle::text_Input(
      inputId = "eventId2",
      label = "Event Name",
      hint_label = "This can be found on the letter",
      error = TRUE
    ),
    # Rich content: a link in the hint
    shinyGovstyle::text_Input(
      inputId = "eventId3",
      label = "Event Name",
      hint_label = shiny::tagList(
        "As shown on your ",
        shinyGovstyle::external_link(
          "https://www.gov.uk",
          "confirmation letter"
        )
      )
    ),
    # Button to trigger error
    shinyGovstyle::button_Input(inputId = "submit", label = "Submit")
  ),
  shinyGovstyle::footer(full = TRUE)
)

server <- function(input, output, session) {
  # Trigger error on blank submit of eventId2
  observeEvent(input$submit, {
    if (input$eventId2 != "") {
      shinyGovstyle::error_off(inputId = "eventId2")
    } else {
      shinyGovstyle::error_on(
        inputId = "eventId2",
        error_message = "Please complete"
      )
    }
  })
```

```
}

if (interactive()) shinyApp(ui = ui, server = server)
```

`transport_data` *Transport data*

Description

Simple example dataset containing transport data, used to demonstrate various table styles in the example app.

Usage

```
transport_data
```

Format

A data frame with 10 rows and 5 variables:

months The month the data belongs to

colours The colour of the vehicle

bikes Cost of bikes

vans Cost of vans

buses Cost of buses

Source

Random data generated for the shinyGovstyle package

`transport_data_small` *Transport data small*

Description

Smaller example dataset containing transport data, used to demonstrate various table styles in the example app.

Usage

```
transport_data_small
```

Format

A data frame with 3 rows and 3 variables:

months The month the data belongs to

bikes Cost of bikes

cars Cost of cars

Source

Random data generated for the shinyGovstyle package

---

update_page_title *Update the browser page title*

Description

Sends a message to the browser to update `document.title` — the text shown in the browser tab. Use this to keep the page title in sync with the content currently on display.

Keeping the page title in sync with the visible page is a GOV.UK Design System recommendation: screen readers announce the title on navigation, and the tab text helps users orient themselves when many tabs are open.

When you need this function: when navigation is triggered programmatically (next / back buttons, footer links pointing to a main page, modal dialogue links) and the visible page changes without the user clicking a nav link directly. Also useful when the page heading differs from the nav link label and you want the tab title to match the heading.

When you don't need this function: when the user clicks a `service_navigation()` link directly and the page title should just match the link text. `service_navigation()` syncs the title automatically by default — see its `auto_page_title` argument.

The JavaScript handler ships with every shinyGovstyle component, so `update_page_title()` works in any app that uses at least one shinyGovstyle function in its UI.

Usage

```
update_page_title(session, page_title, service_name = NULL)
```

Arguments

`session`	The Shiny session object
`page_title`	Character string with the new page title (typically the page heading or nav link text)
`service_name`	Optional character string. When supplied, the browser tab title is set to "`<page_title> | <service_name>`" — the GOV.UK recommended format. When `NULL`, only `page_title` is shown.

Value

NULL, called for side effects

See Also

Other Govstyle navigation: backlink_Input(), navigate_to(), service_navigation(), service_navigation_server(), update_service_navigation()

Examples

```
# Programmatic navigation — keep the page title in sync with the new page.
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::service_navigation(c("Page one", "Page two")),
  bslib::navset_hidden(
    id = "tabs",
    bslib::nav_panel("page_one", shiny::actionButton("next_btn", "Next")),
    bslib::nav_panel("page_two", "Page two content")
  )
)

server <- function(input, output, session) {
  shiny::observeEvent(input$next_btn, {
    bslib::nav_select("tabs", "page_two")
    shinyGovstyle::update_service_navigation(session, "page_two")
    shinyGovstyle::update_page_title(
      session,
      page_title = "Page two",
      service_name = "My service"
    )
  })
}

if (interactive()) shiny::shinyApp(ui = ui, server = server)
```

---

update_radio_button_Input

Update a Govstyle radio button input on the client

Description

Server-side companion to radio_button_Input(), mirroring shiny::updateRadioButtons(). Use it to change the selected option, the available choices, or the label of an existing radio button group from within an observer, for example to keep a cookies settings radio in sync with a choice the user made elsewhere.

Usage

```
update_radio_button_Input(
  session = shiny::getDefaultReactiveDomain(),
  inputId,
  label = NULL,
  choices = NULL,
  selected = NULL,
  inline = FALSE,
  small = FALSE,
  choiceNames = NULL,
  choiceValues = NULL
)
```

Arguments

`session` The `session` object passed to the Shiny server function. Defaults to the current reactive domain.

`inputId` The id of the `radio_button_Input()` to update

`label` New label for the input, or `NULL` to leave unchanged. Sent to the client as plain text: if the group was created with `heading_level` set, the `<hN>` heading wrapper around the legend is preserved, but rich HTML/tag content is not — the legend always ends up as plain text after an update, the same limitation this function has always had.

`choices` New vector of choices, or `NULL` to leave unchanged (if elements of the vector are named then that name rather than the value is displayed to the user)

`selected` The value to select, or `NULL` to leave unchanged

`inline` Whether the radios are inline. Only applies when `choices` (or `choiceNames`/`choiceValues`) are also supplied, since regenerating the options replaces the whole option container; if you do not re-pass it the layout falls back to default. Defaults to FALSE

`small` Whether to use the smaller radios. Only applies when `choices` (or `choiceNames`/`choiceValues`) are also supplied, since regenerating the options replaces the whole option container; if you do not re-pass it the layout falls back to default. Defaults to FALSE

`choiceNames`, `choiceValues` As in `radio_button_Input()`, an alternative to `choices` allowing richer labels. Both must be supplied together

Details

Only the arguments you supply are sent to the client; everything left at its default of `NULL` is left untouched. Because it relies on `session$sendInputMessage()`, the `inputId` is namespaced automatically when called inside a Shiny module, so pass the unnamespaced id just as you would to `shiny::updateRadioButtons()`.

This function is deliberately agnostic about why you are updating the radio, which makes it a useful building block for packages that layer extra behaviour (such as analytics cookie consent) on top of shinyGovstyle. See the cookies and analytics vignette for an example of extending it.

Value

Called for its side effect of sending a message to the client; no return value

See Also

radio_button_Input()

Other Govstyle select inputs: button_Input(), checkbox_Input(), file_Input(), radio_button_Input(), select_Input()

Examples

```
ui <- shinyGovstyle::gov_page(
  shinyjs::useShinyjs(),
  shinyGovstyle::radio_button_Input(
    inputId = "cookies",
    label = "Do you want to accept analytics cookies?",
    choices = c("Yes" = "yes", "No" = "no"),
    selected = "no",
    inline = TRUE
  ),
  shinyGovstyle::button_Input(inputId = "accept", label = "Accept cookies")
)

server <- function(input, output, session) {
  # Keep the radio in sync with a choice made elsewhere
  shiny::observeEvent(input$accept, {
    shinyGovstyle::update_radio_button_Input(
      session,
      inputId = "cookies",
      selected = "yes"
    )
  })
}
if (interactive()) shinyApp(ui = ui, server = server)
```

---

update_service_navigation

Update the active item in a service navigation component

Description

Sends a message to the browser to update the highlighted active item in the service navigation bar.

When you need this function: when navigation is triggered programmatically — for example, via a next / back button or a footer link that points to a main page. In those cases the nav link itself is not clicked, so the JavaScript binding does not fire and the active state does not update automatically. Call `update_service_navigation()` alongside your tab-switching call to keep them in sync.

When you don't need this function: when the user clicks a service navigation link directly. The JavaScript binding updates the active state automatically, so you only need to switch the tab panel in your `observeEvent()`.

`inputId` is cleaned up the same way as in `service_navigation()`, so you can pass either the inputID you supplied there (e.g. `"Page-Two"`) or its cleaned form (`"page_two"`).

Usage

```
update_service_navigation(session, inputId)
```

Arguments

`session`	The Shiny session object
`inputId`	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via `input$<inputId>`.

Value

NULL, called for side effects

See Also

Other Govstyle navigation: `backlink_Input()`, `navigate_to()`, `service_navigation()`, `service_navigation_server()`, `update_page_title()`

Examples

```
# Works the same with shiny::tabsetPanel() + shiny::updateTabsetPanel().
ui <- shinyGovstyle::gov_page(
  shinyGovstyle::service_navigation(c("Page one", "Page two")),
  bslib::navset_hidden(
    id = "tabs",
    bslib::nav_panel("page_one", shiny::actionButton("next_btn", "Next")),
    bslib::nav_panel("page_two", "Page two content")
  )
)

server <- function(input, output, session) {
  # Nav link clicked - JS handles the active state, just switch the panel.
  shiny::observeEvent(input$page_two, {
    bslib::nav_select("tabs", "page_two")
  })

  # Programmatic navigation - the nav link is not clicked, so call
  # update_service_navigation() explicitly.
  shiny::observeEvent(input$next_btn, {
    bslib::nav_select("tabs", "page_two")
    shinyGovstyle::update_service_navigation(session, "page_two")
  })
}
```

```
if (interactive()) shiny::shinyApp(ui = ui, server = server)
```

`value_box` *Create a value text box with optional description and colour*

Description

This function generates a value text box with an optional description and customisable colour.

Usage

```
value_box(value = "your value goes here", text = NA, colour = "blue")
```

Arguments

`value`	Character. The primary value to display in the value box. Defaults to "your value goes here"
`text`	Character or NA. An optional description to appear below the value. If not provided (default is NA), the description will not be displayed
`colour`	Character. A colour to apply to the value box. Defaults to "blue". Choose from the following: "grey", "purple", "teal", "blue", "yellow", "orange", "red", "magenta", or "green". Any other value gives a warning listing these options

Details

The text box can be used in Shiny applications to display highlighted information, such as statistics or key metrics.

Value

A Shiny `div` tag representing the value box, styled according to the specified parameters

See Also

Other Govstyle feedback types: `confirmation_panel()`, `details()`, `gov_summary()`, `gov_tag()`, `inset_text()`, `label_hint()`, `noti_banner()`, `warning_text()`

Examples

```
value_box(
  value = "1,000,000",
  text = "This is the latest value for the selected inputs.",
  colour = "purple"
)
```

warning_text *Warning Text Function*

Description

This function create warning text.

Usage

```
warning_text(inputId, text)
```

Arguments

inputId	The id assigned to the component's root element. For Shiny input components this is also the name used to access the value via input$<inputId>.
text	Text that goes in the main. Accepts a plain character string, or shiny tag objects such as shiny::tags$b("Bold") or a shiny::tagList().

Value

a warning box HTML shiny tag object

See Also

Other Govstyle feedback types: confirmation_panel(), details(), gov_summary(), gov_tag(), inset_text(), label_hint(), noti_banner(), value_box()

Examples

```
shinyGovstyle::warning_text(
  inputId = "warn1",
  text = "You can be fined up to £5,000 if you do not register."
)
```

Index

∗ **Govstyle actions**
download_button, 16
download_link, 18
download_radios, 19
download_radios_handler, 21
external_link, 27
∗ **Govstyle errors**
error_off, 22
error_on, 23
error_summary, 24
error_summary_update, 26
∗ **Govstyle feedback types**
confirmation_panel, 10
details, 15
gov_summary, 44
gov_tag, 45
inset_text, 50
label_hint, 51
noti_banner, 56
value_box, 78
warning_text, 79
∗ **Govstyle navigation**
backlink_Input, 5
navigate_to, 54
service_navigation, 64
service_navigation_server, 66
update_page_title, 73
update_service_navigation, 76
∗ **Govstyle page structure**
cookieBanner, 12
footer, 32
gov_page, 42
header, 46
layouts, 53
phase_banner, 57
skip_to_main, 67
∗ **Govstyle select inputs**
button_Input, 7
checkbox_Input, 8
file_Input, 29
radio_button_Input, 59
select_Input, 62
update_radio_button_Input, 74
∗ **Govstyle styling**
font, 31
∗ **Govstyle tables tabs and accordions**
accordion, 3
govReactable, 35
govReactable-shiny, 37
govTable, 39
govTabs, 40
∗ **Govstyle text types**
date_Input, 13
gov_list, 41
heading_text, 48
input_field, 48
text_area_Input, 68
text_Input, 70
∗ **datasets**
bad_link_text, 6
case_data, 8
transport_data, 72
transport_data_small, 72

accordion, 3
accordion(), *36*, *38–40*

backlink_Input, 5
backlink_Input(), *55*, *65*, *66*, *74*, *77*
bad_link_text, 6
button_Input, 7
button_Input(), *9*, *30*, *60*, *63*, *76*

case_data, 8
checkbox_Input, 8
checkbox_Input(), *7*, *30*, *60*, *63*, *76*
confirmation_panel, 10
confirmation_panel(), *16*, *44*, *46*, *51*, *52*, *57*, *78*, *79*

cookieBanner, 12
cookieBanner(), *33*, *43*, *47*, *53*, *58*, *68*

date_Input, 13
date_Input(), *41*, *48, 49*, *69, 70*
details, 15
details(), *11*, *44*, *46*, *51, 52*, *57*, *78, 79*
download_button, 16
download_button(), *19–21*, *28*
download_link, 18
download_link(), *17*, *20, 21*, *28*
download_radios, 19
download_radios(), *17*, *19*, *21*, *28*
download_radios_handler, 21
download_radios_handler(), *17*, *19, 20*, *28*

error_off, 22
error_off(), *23*, *25, 26*
error_on, 23
error_on(), *22*, *25, 26*
error_summary, 24
error_summary(), *22, 23*, *26*
error_summary_update, 26
error_summary_update(), *22, 23*, *25*
external_link, 27
external_link(), *17*, *19–21*

file_Input, 29
file_Input(), *7*, *9*, *60*, *63*, *76*
font, 31
footer, 32
footer(), *12*, *43*, *47*, *53*, *58*, *68*

gov_box *(*layouts*)*, 53
gov_list, 41
gov_list(), *14*, *48, 49*, *69, 70*
gov_main_layout *(*layouts*)*, 53
gov_page, 42
gov_page(), *12*, *33*, *47*, *53*, *58*, *68*
gov_row *(*layouts*)*, 53
gov_summary, 44
gov_summary(), *11*, *16*, *46*, *51, 52*, *57*, *78, 79*
gov_tag, 45
gov_tag(), *11*, *16*, *44*, *51, 52*, *57*, *78, 79*
gov_text *(*layouts*)*, 53
govReactable, 35
govReactable(), *3*, *38–40*
govReactable-shiny, 37
govReactableOutput
*(*govReactable-shiny*)*, 37
govTable, 39
govTable(), *3*, *36*, *38*, *40*
govTabs, 40
govTabs(), *3*, *36*, *38, 39*

header, 46
header(), *12*, *33*, *43*, *53*, *58*, *68*
heading_text, 48
heading_text(), *14*, *41*, *49*, *69, 70*
htmltools::tags(), *28*

input_field, 48
input_field(), *14*, *41*, *48*, *69, 70*
inset_text, 50
inset_text(), *11*, *16*, *44*, *46*, *52*, *57*, *78, 79*

label_hint, 51
label_hint(), *11*, *16*, *44*, *46*, *51*, *57*, *78, 79*
layouts, *12*, *33*, *43*, *47*, 53, *58*, *68*

navigate_to, 54
navigate_to(), *5*, *65, 66*, *74*, *77*
noti_banner, 56
noti_banner(), *11*, *16*, *44*, *46*, *51, 52*, *78, 79*

phase_banner, 57
phase_banner(), *12*, *33*, *43*, *47*, *53*, *68*

quote(), *38*

radio_button_Input, 59
radio_button_Input(), *7*, *9*, *30*, *63*, *74–76*
renderGovReactable
*(*govReactable-shiny*)*, 37
run_example, 62

select_Input, 62
select_Input(), *7*, *9*, *30*, *60*, *76*
service_navigation, 64
service_navigation(), *5*, *55*, *66*, *74*, *77*
service_navigation_server, 66
service_navigation_server(), *5*, *55*, *65*, *74*, *77*
shiny::checkboxGroupInput(), *60*
shiny::updateRadioButtons(), *74, 75*
skip_to_main, 67
skip_to_main(), *12*, *33*, *43*, *47*, *53*, *58*

text_area_Input, 68
text_area_Input(), *14*, *41*, *48, 49*, *70*

text_Input, 70
text_Input(), *14*, *41*, *48*, *49*, *69*
transport_data, 72
transport_data_small, 72

update_page_title, 73
update_page_title(), *5*, *55*, *65*, *66*, *77*
update_radio_button_Input, 74
update_radio_button_Input(), *7*, *9*, *30*, *60*, *63*
update_service_navigation, 76
update_service_navigation(), *5*, *55*, *64–66*, *74*

value_box, 78
value_box(), *11*, *16*, *44*, *46*, *51*, *52*, *57*, *79*

warning_text, 79
warning_text(), *11*, *16*, *44*, *46*, *51*, *52*, *57*, *78*